A model for visual building SPARQL queries

Fabio Benedetti, Sonia Bergamaschi

Dipartimento di Ingegneria "Enzo Ferrari" - Università di Modena e Reggio Emilia - Italy
firstname.lastname@unimore.it

Abstract. LODeX is a Semantic Web tool that, leveraging a summarized representation of a LOD source structure (i.e. Schema Summary), helps users explore and query SPARQL endpoints by hiding the complexity of Semantic Web technologies. By leveraging Schema Summary of a LOD source, LODeX guides the user in composing visual queries that are automatically translated in correct SPARQL queries through a SPARQL compiler. In this work we inspected how LODeX can deal with the high expressivity of SPARQL. In particular, we propose a formal model that allow to define queries over the Schema Summary (i.e. Basic Query) and we analyze how this model can handle different join patterns used in SPARQL queries. Finally, we inspect how LODeX can satisfy real world users necessities by analyzing the query logs contained in the LSQ dataset. We show that LODeX could be able to generate the 77.6% of the 5 million queries contained in LSQ dataset.

1 Introduction

In the last few years, the Linked Data Cloud is growing at dizzying pace, its number of datasets has risen from 294 in the 2009 to 1024 in the 2014 [23], without taking in consideration the huge amount of Semantic data recently embedded in web pages thanks to the RDFa standard [2]. In this context, one of the greatest issue is the complexity of the Semantic Web technologies, indeed, a user for effectively dealing with this kind of data have to master several complex standards: RDF [10], a metadata data model used for defining semantic data according to a graph data model; RDFS [8] and OWL [16], languages for defining ontological constraints and enabling inference upon semantic data; SPARQL [24], a query language used for extracting information from RDF data. Moreover, each data sets is usually described by its own vocabulary, increasing on one hand the flexibility in data representation but raising on the other hand the complexity in data consumption, especially for unskilled users. Writing SPARQL queries is a error-prone and tedious task but today is the only way for extracting information from RDF data.

For supporting the user in the consumption of Linked Data we defined a model, called Schema Summary [5] [7], and a tool, called LODeX [4] [6], both of them finalized in supporting the user in exploring an unknown datasets. Each LOD source is composed by two kind of knowledge: the *intensional knowledge* defines the terminology used in the dataset and it is described in RDFS or OWL; the *extensional knowledge* usually covers most of the datasets and it contains the entities of the real world described in the dataset. The Schema Summary of a LOD source is extracted by analyzing the distribution of the instances within the extensional knowledge and LODeX allows user to build visual queries over this summary. In the paper we want to inspect how much the expressivity of SPARQL is covered by the SPARQL queries that can be generated by LODeX. In particular, we propose the definition of the *Basic query* model used within LODeX and the algorithm of the SPARQL compiler, that allow to generate SPARQL queries starting from the Schema Summary of a LOD source and an instance of Basic Query. Then, we inspect how the Basic Query model can handle different join pattern that can be found in SPARQL. Finally, we evaluate how LODeX can satisfy the real world users requests by using the LQS Dataset [21]. This dataset contains more than 5 million of queries, extracted from logs of 3 different SPARQL endpoints, executed from real users. The analysis that we propose is finalized to verify how many of these query could be generated by LODeX.

The remainder of the paper is structured as follows. We discuss related work in Section 2. We summarize the architecture of LODeX in Section 3. Section 4 illustrates the component of LODeX that manage the formulation and translation of queries, while Section 5 reports the capabilities and limitations of the Basic Query model. Section 6 contains the analysis of the queries contained in LSQ respect to the capabilities of LODeX, and finally, Section 7 sketches the conclusions.

2 Related Works

In the literature, we can find several tools that aim to support users in LOD visualization, browsing and in the definition of complex queries. These tools can be divided in two major groups: tools with the main goal of providing a synthetic overview of the whole structure of an RDF dataset and tools that allow the browsing of a dataset just at instance level.

In the first group, we can find only few examples; the most important two are LOD Visualization and ProLOD; these tools aim to provide to users an high level analysis of a LOD dataset. In particular, LOD Visualization is a prototype based on the Linked Data Visualization Model [9], and it allows to build analysis, transformations and visualizations of Linked Data. Whereas, ProLOD [1] automatically provides a group of statistical analysis regarding the content of a dataset, but it does not foresee any querying possibility.

In the second group we can find tools that provide visual querying functionalities, but their focus is limited to the instance level. A first example is LD Query Wizard [15], a tool that allows to select instances through keyword search and it uses a powerful tabular view that allows users to browse an RDF graph. LODlive and LODmilla [17] provide a visually appealing way to explore information associated with an instance using a graph visualization, but they do not provide any query building functionality. Also gFacet [13, 14] uses the same strategy of exploration (with a graph visualization), but in this case, each node is a class that contains a list of instances and the user can link new nodes (classes) as if he was building a visual query, but it does not produce as output any SPARQL query. SPARKLIS [12] implements a fascinating approach in which a SPARQL query is composed as if the user was composing a natural language request. ISPARQL [18] is one of the most famous tool for building visual SPARQL query and it allows to incrementally build a SPARQL query by extending it step by step; this kind of approach is not very intuitive and a user have to own a good knowledge of SPARQL for being able to use ISPARQL. Moreover, the tool does not provide any guidance to users, indeed, they have to understand the schema of the LOD source before being able to define a SPARQL query.

As reported in [11], the majority of the tools for data visualization requires the user to manually explore the dataset and they are not able to provide a synthetic *schema* representing the structure of the source. LODeX differs from the tools described above since it provides a synthetic representation of LOD source schema and the user can use it to build visual queries. Moreover, LODeX hides to the final user the complexity of the Semantic Web technologies allowing users to build SPARQL query without knowing its syntax. The only tool that provide this characteristic is SPARKLIS which, however, does not provide user a whole picture of the structure of the dataset.

3 Architecture Overview

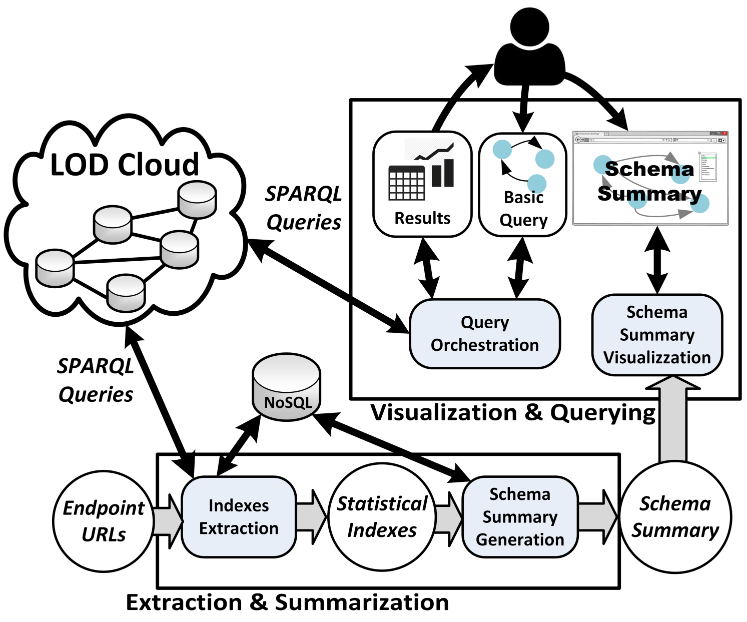

Fig. 1: LODeX architecture

LODeX consists of four distinct components (as illustrated in Figure 1), each responsible for a specific activity:

- **Indexes Extraction**: it takes as input the URL of a SPARQL endpoint and generates a set of queries able to extract a set of indexes from the extensional knowledge (extensional group of Statistical Indexes in [3]).

- **Schema Summary Generation**: it generates the Schema Summary thanks to the indexes extracted in the previous step (see [7] for additional information).
- **Schema Summary Visualization**: the visualization is performed by a web application through which the user can interact for browsing the Schema Summary and build the visual query.
- **Query Orchestration**: it manages the interaction between the GUI and the user in composing the visual query, in the generation of the SPARQL queries and in the submission of it to the remote endpoint.

4 Query Orchestration

The Query Orchestrator is the component that manages the formulation of the visual queries and the translation of these queries in SPARQL ones. These functionalities are available thanks to a model, called *Basic Query*, used for describing the visual query that the user is building. A SPARQL compiler takes in input an instance of a *Basic Query* and the related Schema Summary for producing as output the corresponding SPARQL query.

4.1 Basic Query

The user is guided by LODex in the composition of a Basic Query upon a Schema Summary. A Schema Summary is a pseudo-graph where the nodes are classes connected through properties; each classes is associated to a set of attributes, each attribute of a class represent the existence of a particular datatype property having as subject an instance of this class. A formal definition of the Schema Summary and a description of the interface of LODeX can be found in [7]. The classes, properties and attributes selected by the user in Schema Summary participate to the composition of a Basic Query Q. A Basic Query has a tree structure that overlaps the Schema Summary graph, the nodes of the tree are classes $\in C$, while the leafs can be both classes $\in C$ or attributes $\in A$. Here is its formal definition:

Definition 1 (Basic query) *A basic query Q, defined on a schema summary S, is a tuple Q= (T,m_C,m_E,m_A,o,F), where*

- *T is a directed and labeled tree composed by two kinds of vertices, T = (C',A',E'), where:*
  - *C' is a set of vertices where each vertex refers to a class in the Schema Summary*
  - *A' is a set of vertices where each vertex refers to an attribute in the Schema Summary. This kind of node can appear only as leaf of the tree T*
  - *E' is a set of edges where each edge consist of an ordered pair of vertices e=<n1,n2>, e ∈ E', n1 ∈ C', n2 ∈ C' ∪ A'*
  - *r ∈ C' is the root node,*
- $m_C : C' \to C$ *is a mapping function that links each vertex in C' with a vertex in C(S);*

- $m_A\colon A' \rightarrow A$ *is a mapping function that links each vertex in A' with an edge in A(S);*
- $m_E\colon E' \rightarrow P$ *is a mapping function that links each edge in E' with an edge in P(S);*
- $o\colon E' \rightarrow \{true, false\}$ *is called optionality function.*
- *F contains the filtering conditions associated with the attributes of the query and it is composed by a set of tuple (a,k,v), where a is the attribute* $\in$ *A', k contains the filtering operator (a regular expression, or an arithmetic operator) and v is the value through which to assess the filter expression.*

Graphically, a user starts composing a basic query by selecting the first class in the Schema Summary, then, if the user selects a property for this first class, also the connected class is shown in the query panel and the edge and vertex are added to the tree. The user may also select the attributes of each class: in this case, the tree is further enriched with edges and leafs.

4.2 SPARQL compiler

The Query Orchestrator translates the Basic Query into a SPARQL query through a compiler. The compiler exploits an iterative algorithm that traverses the Basic Query tree for producing the SPARQL query. The Query Orchestrator allows the usage of these SPARQL operators: AND (.), OPTIONAL (also nested), FILTER, ORDER BY, OFFSET and LIMIT. The pseudo-code of the SPARQL compiler is presented in Algorithm 1. For simplicity, I show only the compilation of the body of the query, then, this body will be equipped with the SELECT statement, the pagination condition (LIMIT and OFFSET) and the ORDER condition, if specified. Algorithm 1 takes as input the Schema Summary S and the basic query Q. The output (SQ) of Algorithm 1 contains a SPARQL query. I use some of the algebraic operators defined in [19] to describe the formulation of the query: the triple pattern *(s,p,o)*, in which the elements can also be parameters (in this case we use the function $Par(node)$ for assigning a unique parameter); the operator *AND* that represents the dot (.) in SPARQL; the operator *OPT* (OPTIONAL); the operator *FILTER* (i.e. ?parameter condition value). The compiler is based on a recursive algorithm which is initialized from line 2 to 4, where the the BGPs (i.e. basic graph pattern [20]) related to the root node of the Basic Query are translated. The function m_C, m_A and m_E retrieve the URIs related to classes, attribute and properties from the Schema Summary S. The function $attributes(class, currentquery)$ append to $currentquery$ the statements related to the attributes of the current class. The function $filtCon(a)$ returns the filter statements associated to the current attribute. When the algorithm is initialized, the recursive function $recursivebody(class, currentquery)$ is called (line 4); this function traverse the query tree and it inserts the statements regarding the various nodes. The edges composing the query can refers to properties that exist in the Schema Summary with a revers direction, this case is handled in lines 10 and 11. When the algorithm has traversed all the query tree the SPARQL query is returned.

```
Data: S, Q, F
Result: SQ
Algorithm compiler()
    SQ=($Par$(Q.r),rdf:type,$m_C$(Q.r));
    SQ=attributes(Q.r,SQ);
    SQ=recursivebody(Q.r,SQ);
    return SQ;
Function recursivebody(c,RQ)
    forall the child ch of c do
        CHQ=($Par$(ch),rdf:type,$m_C$(ch));
        if $m_E$(<c,ch>) is directed from c to ch in S then CHQ=CHQ AND
        ($Par$(c),$m_E$(<c,ch>),$Par$(ch));
        else CHQ=CHQ AND ($Par$(ch)),$m_E$(<c,ch>),$Par$(c);
        CHQ=attributes(ch,CHQ);
        if ch is not a leaf node then recursivebody(ch,CHQ);
        if o(ch) then RQ=RQ OPT ( CHQ );
        else RQ=RQ AND CHQ;
    end
    return RQ ;
Function attributes(c,SQ)
    forall the attribute a connected with c do
        AQ=($Par$(c),$m_A$(<c,a>),$Par$(a));
        if exist F(a) then AQ=AQ AND filtCon (a);
        if o(a) then SQ=SQ OPT ( AQ );
        else SQ=SQ AND AQ;
    end
    return SQ ;
Function filtCon(a)
    FC=$\emptyset$;
    forall the fc in Q.F related to a do
        concatenate in FC, using the AND operator, each fc condition as: FILTER
        ($Par$(a), fc.k fc.v);
    end
    return FC ;
```

Algorithm 1: SPARQL compiler algorithm

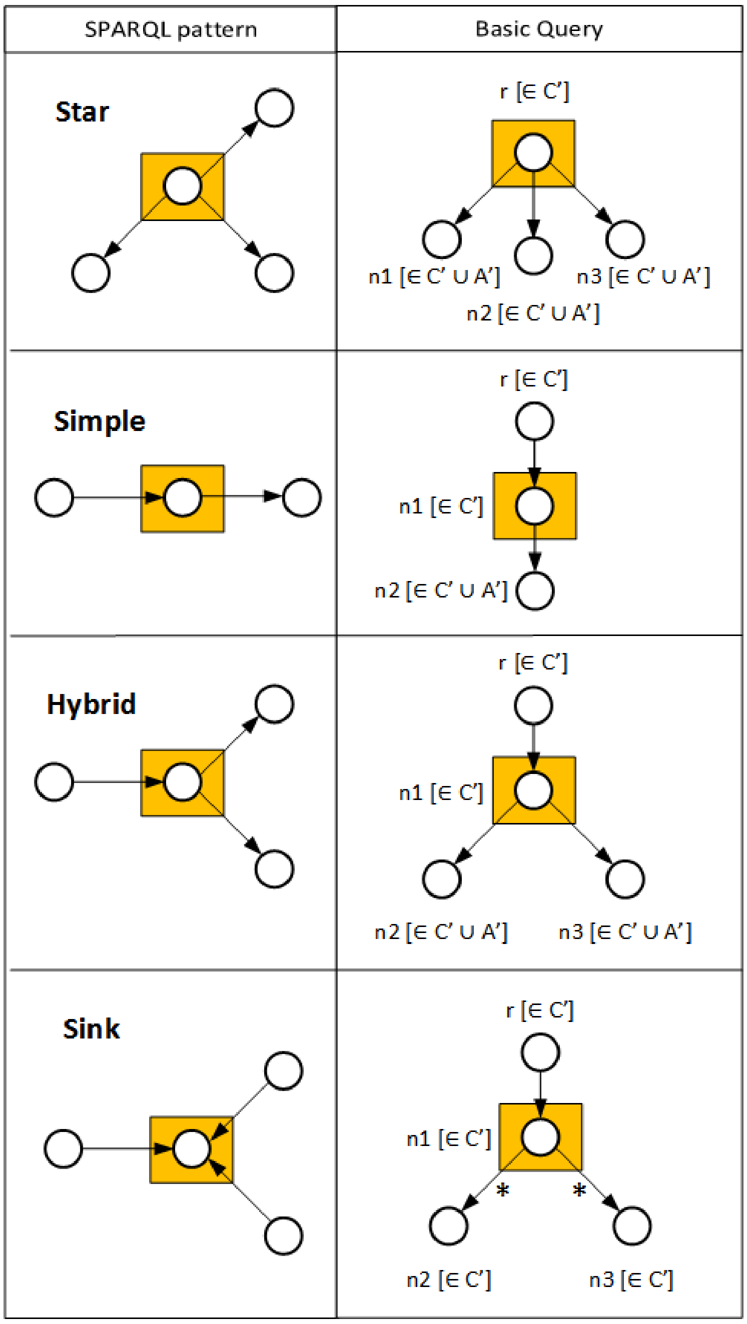

Fig. 2: Kinds of hypergraph patterns and their representation as Basic Query (Legend: *r* root node, *C'* class, *A'* attribute, * the Basic Query is translated in a SPARQL query and the direction of the edge is reversed).

5 Capabilities and limitations of the Basic Query model

The authors in [22] defined 4 different kinds of join types that can be found in a SPARQL query. These join patterns differ in the way the join node is connected within the BGPs forming the SPARQL query. A join node has an *outgoing link* if it appears as subject of a triple pattern and it has an *incoming link* if it appears as object. In a **Star** pattern the join node has multiple incoming links but no outgoing links. A **Simple** pattern contains a join node that has precisely one incoming and one outgoing link. A **Hybrid** pattern is composed by three ore more links and the join node has at least one incoming and one outgoing link. Finally, a **Sink** pattern contains a join node with multiple incoming links and no outgoing ones. In Figure 2 on the left you can see a schematic representation of these four patterns, while on the right their representation through the Basic Query model is proposed. As you can see they can all be represented through the Basic Query model and, even if the *Sink* pattern expects the existence of only incoming links, the SPARQL compiler can translate edges between nodes of the query in the opposite direction, according to the information contained in the Schema Summary. The Basic Query is able to model all the acyclic combinations of these pattern. A tree model is not enough expressive to represent cyclic queries, but this kind of queries are not so common in real use cases (as it will be shown in Section 6).

From the point of view of the SPARQL operators supported, LODeX is able to generate SPARQL queries with a SELECT form. The OPTIONAL pattern is allowed, indeed, the user can include optional attributes and classes in the Basic Query and the SPARQL compiler is able to correctly translate the query; if a class is included in the query as optional, all its child have to be included in the OPTION clause. LODeX supports the FILTER operator that can be applied to attributes (the greatest part FILTER clauses are applied on literal values). In particular, LODeX allow the definition of filter statement on number (greater, minor or equal) and string (equal, contains and regular expression). LODeX allows also the usage of the operator ORDER BY and it can be used with both classes and attributes. Moreover, the operators LIMIT and OFFSET are available, indeed, LODeX automatically paginates the results in order to speed up the retrieval of results.

LODeX does not implement all the SPARQL 1.1 operators; in particular, UNION, GROUP BY, MINUS, EXCEPT are not available. This choice is primarily due to the willing of providing an intuitive visual query interface, and the usage of such operators would have increased the complexity in the query definition. In fact, LODeX aims to be a tool used for exploring the content of an unknown LOD source and for easily building simple queries. Moreover, LODeX allows the user to manually modify the generated SPARQL query and a skilled user, who master these complex operators, can quickly build the skeleton of his query with LODeX and then modify it at will.

6 Evaluation using real world query logs

The LSQ datasets contains SPARQL queries extracted from endpoint logs [1] executed on three public datasets: DBpedia (logs from 30/04/201020/07/2010; a dataset with

[1] The LSQ dataset is available from `http://aksw.github.io/LSQ/`.

232 million triples), Linked Geo Data (LGD) (24/11/2010 06/07/2011; with 1 billion triples), Semantic Web Dog Food (SWDF) (16/05/201412/11/2014; with 300 thousand triples) and the British Museum (BM) (08/11/201401/12/2014; with 1.4 million triples). These queries represent a good sample for evaluating the real usage of SPARQL and we analyzed them for discovering how much the limitations of the Basic Query model would affect a real scenario.

Table 1: Number of queries contained in the LSQ dataset and number of queries involving the intensional or extensional knowledge.

Dataset	**Total Queries**	**Distinct Queries**	**Correct Queries**	**Intensional Knowledge**	**Extensional Knowledge**	**% Extensional**
DBpedia 3.5.1	1728041	1208789	782364	43812	738552	94.4
Linked Geo Data	1656254	311126	297580	4761	292819	98.4
SWDF	1411483	99165	85520	4533	80987	94.7
British Museum	879426	129989	29073	0	29073	100
Total	5675204	1749069	1194537	53106	1141431	**95.55**

The number of queries contained in the dataset is more than 5 million, but, by removing the duplicate queries and the malformed ones the number goes down to 1.2 million (as you can see in the left part of Table 1). The Schema Summary produced by LODeX covers only the extensional knowledge of a LOD source [5]. Thus, we first inspected which portion of the queries contain some BGPs targeting the intensional knowledge. The results are shown in the right part of Table 1; as you can see just the 4.45% of the queries contains some clauses involving the intensional knowledge. This is not a surprising result because only users with a deep knowledge about Semantic Web usually target this kind of data.

Table 2: Percentage of queries containing join nodes and cyclic queries

Dataset	**Star**	**Path**	**Hybrid**	**Sink**	**No Join**	**Cyclic**
DBpedia3.5.1	38.57	8.6	6.79	6.31	61.23	4.26
Linked Geo Data	28.18	9.46	7.57	1.24	72	2.58
SWDF	10.7	11.24	4.01	0.92	84.25	4.11
British Museum	0	0	0	0	100	0
Overall	33.05	8.79	6.62	4.51	66.5	**3.72**

Secondly, we analyzed the queries looking for what kinds of join structure they contain. In Table 2 you can see a summary of the results. The greatest part of the queries have a very basic structure, with no join, or with a single join node. Only the 3.72% of the queries contains a combination of join nodes that forms cycle and they are not compatible with the Basic Query model.

We also analyzed the operators used. In Table 3 the utilization percentage of the principal operators not supported by LODeX is summarized. As you can see in the top Table, advanced features like federation, sub-query and negation are rarely used. Aggregation features (GROUP BY) are a bit more used but they are lower long the 1%. The bottom Table contains others operators: UNION, CONSTRUCT, DESCRIBE and ASK. In this case the impact is a bit higher but two of the operators (DESCRIBE and ASK) are used primarily for exploring a dataset when a user do not know its structure. LODeX do not need these operators because the structure of source is revealed thanks to the Schema Summary. The most impactful operator is UNION that involves the 8% of the queries overall and almost the 33% in the SWDF dataset.

Table 3: Percentage of queries containing unsupported operators.

Dataset	**Federation**	**Sub-Query**	**Aggregators**	**Negation**
DBpedia3.5.1	0.0005	0.0037	0.0005	0.001
Linked Geo Data	0	0.0074	0.0074	0
SWDF	0.0012	0.0152	2.4053	0.0012
British Museum	0	0	0	0
Total	0.0004	0.0054	0.1744	0.0008

Dataset	**UNION**	**COSNTRUCT**	**DESRIBE**	**ASK**
DBpedia3.5.1	4.4153	0.8991	0.0977	4.3572
Linked Geo Data	9.6475	2.3115	0.0081	8.3676
SWDF	32.714	0.0493	31.1165	0.0645
British Museum	0	0	0	0
Total	7.6373	1.1684	2.2881	4.9438

In Figure 3 a summary of this analysis is shown. As you can see LODeX could support user in the definition of the greatest part of the unique SPARQL queries contained in LSQ. In particular, the global percentage of executable queries is 77.79%. We obtained the worst results with the SWDF endpoints (56.75% of queries executable) and the main reason is the kind of users that usually queries this dataset. In fact, SWDF is a dataset containing data regarding conferences and articles about the Semantic Web, so, the most probable kind of consumers are academics and Semantic Web experts. The Semantic Web skill of these users is above the average and they are outside the possible range of users that could use LODeX for building their SPARQL queries.

7 Conclusion

The Basic Query model represent an expressive model that can satisfy real world needs in supporting users in formulating SPARQL queries. LODeX, that internally uses this model, represents a powerful tool in supporting users in exploring unknown LOD sources [7]. In this paper we showed that LODeX could be able to support users in the formulation of the 77.6% of the queries contained in LSQ dataset. Some of the limitations of LODeX, like the focus in the extensional knowledge, and of the Basic Query model, like the impossibility of composing cyclic queries, have barely affected this result. However,

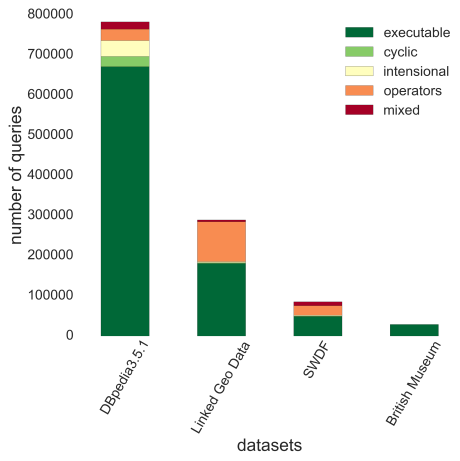

Fig. 3: Summary of the number of unique queries executable or not through LODeX. Legend: *executable*: executable through LODeX; *cyclic*: cyclic query not executable through LODeX; *intensional*: queries not executable through LODeX because they target the intensional knowledge; *operators*: queries containing unsupported SPARQL operators; *mixed*: unsupported queries for more than one specific reason of the latter three.

this analysis allows to delineate future lines of improvement. First of all, we could ulteriorly meet users needs by including the possibility of building UNION queries. Successively, we could focus our effort in extending the Basic Query model for enabling aggregation queries.

References

1. Z. Abedjan, T. Grütze, A. Jentzsch, and F. Naumann. Profiling and mining RDF data with prolod++. In I. F. Cruz, E. Ferrari, Y. Tao, E. Bertino, and G. Trajcevski, editors, *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 1198–1201. IEEE, 2014.
2. B. Adida, M. Birbeck, S. McCarron, and S. Pemberton. Rdfa in xhtml: Syntax and processing. *Recommendation, W3C*, 7, 2008.
3. F. Benedetti, S. Bergamaschi, and L. Po. Online index extraction from linked open data sources. *Linked Data for Information Extraction (LD4IE) Workshop held at International Semantic Web Conference*, 2014.
4. F. Benedetti, S. Bergamaschi, and L. Po. A visual summary for linked open data sources. *International Semantic Web Conference (Posters & Demos) 2014*, 2014.
5. F. Benedetti, S. Bergamaschi, and L. Po. Exposing the underlying schema of lod sources. In *Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2015 IEEE/WIC/ACM International Joint Conferences*, 2015.
6. F. Benedetti, S. Bergamaschi, and L. Po. Lodex: A tool for visual querying linked open data. In *International Semantic Web Conference (ISWC 2015 Posters & Demos)*, CEUR Workshop Proceedings. CEUR-WS.org, 2015.

7. F. Benedetti, S. Bergamaschi, and L. Po. Visual querying lod sources with lodex. In *Proceedings of the 8th International Conference on Knowledge Capture*, K-CAP 2015, pages 12:1–12:8, New York, NY, USA, 2015. ACM.
8. D. Brickley and R. V. Guha. {RDF vocabulary description language 1.0: RDF schema}. 2004.
9. J. M. Brunetti, S. Auer, and R. Garca. The linked data visualization model. In *International Semantic Web Conference (Posters & Demos)*, 2012.
10. R. Cyganiak, D. Wood, and M. Lanthaler. Rdf 1.1 concepts and abstract syntax. *W3C Recommendation*, 25:1–8, 2014.
11. A.-S. Dadzie and M. Rowe. Approaches to visualising linked data: A survey. *Semantic Web*, 2(2):89–124, 2011.
12. S. Ferré. Expressive and scalable query-based faceted search over sparql endpoints. In P. Mika, T. Tudorache, A. Bernstein, C. Welty, C. Knoblock, D. Vrandei, P. Groth, N. Noy, K. Janowicz, and C. Goble, editors, *The Semantic Web ISWC 2014*, volume 8797 of *Lecture Notes in Computer Science*, pages 438–453. Springer International Publishing, 2014.
13. P. Heim, T. Ertl, and J. Ziegler. Facet graphs: Complex semantic querying made easy. In *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part I*, pages 288–302, 2010.
14. P. Heim, J. Ziegler, and S. Lohmann. gfacet: A browser for the web of data. In *Proceedings of the International Workshop on Interacting with Multimedia Content in the Social Semantic Web (IMC-SSW'08) Koblenz, Germany, December 3, 2008.*, 2008.
15. P. Höfler, M. Granitzer, E. E. Veas, and C. Seifert. Linked data query wizard: A novel interface for accessing SPARQL endpoints. In *Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014), Seoul, Korea, April 8, 2014.*, 2014.
16. D. L. M. P. F. P.-S. Jie Bao, Elisa F. Kendall. Owl 2 web ontology language quick reference guide, Oct. 2009.
17. C. Kiefer and A. Bernstein. Lodmilla: a linked data browser for all. In *Proceedings of the Posters and Demos Track of 10th International Conference on Semantic Systems*, SEMANTiCS2014, pages 31–34. CEUR-WS.org, 2014.
18. C. Kiefer, A. Bernstein, and M. Stocker. The fundamentals of isparql: A virtual triple approach for similarity-based semantic web tasks. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 295–309, Berlin, Heidelberg, 2007. Springer-Verlag.
19. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):16, 2009.
20. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):1–45, Aug. 2009.
21. M. Saleem, M. I. Ali, A. Hogan, Q. Mehmood, and A.-C. N. Ngomo. Lsq: The linked sparql queries dataset. In *The Semantic Web-ISWC 2015*, pages 261–269. Springer, 2015.
22. M. Saleem and A.-C. N. Ngomo. Hibiscus: Hypergraph-based source selection for sparql endpoint federation. In *The Semantic Web: Trends and Challenges*, pages 176–191. Springer, 2014.
23. M. Schmachtenberg, C. Bizer, and H. Paulheim. Adoption of the linked data best practices in different topical domains. In *The semantic web–ISWC 2014*, pages 245–260. Springer, 2014.
24. A. S. Steve Harris. Sparql 1.1 query language, Mar. 2013.